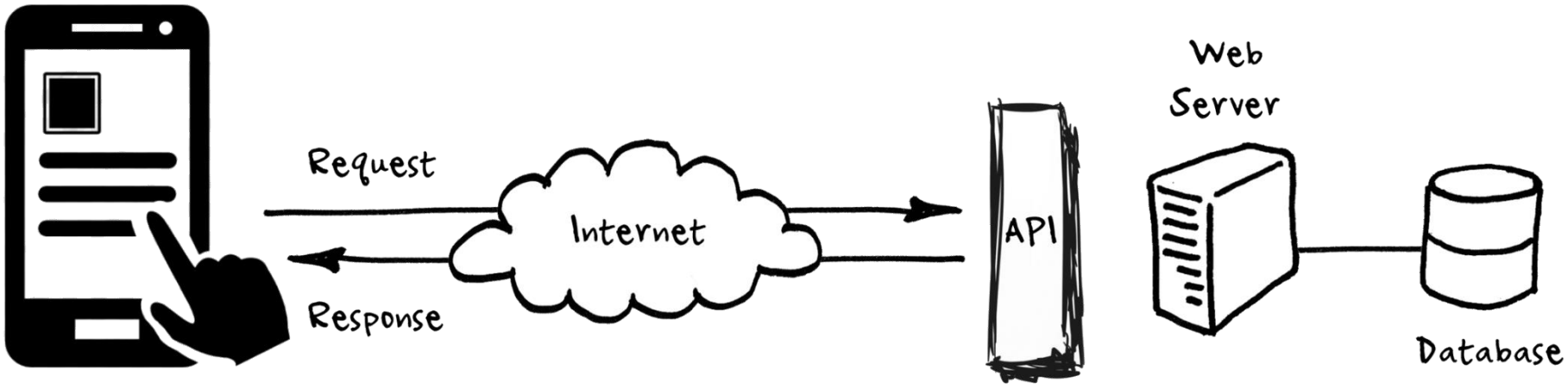
Calling Web API using Coroutines



Outline

1. Web API
2. Accessing Web API using Ktor and Coroutines

Web API
(aka Web Services / REST API)

Working with Web APIs – the Why?

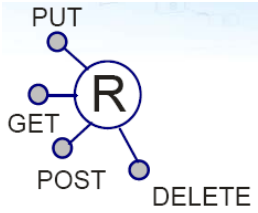
- Phones can not serve as centralized data stores, so we **need servers**
- Even when we can do heavy tasks on-device, we should not
  - Servers are powerful, phones are not
  - Processing a lot of data / complex computation on a phone is a drain on its resources: Battery, CPU, Memory
- As good citizens on an Android phone, our **apps should consume as little resources as possible**
- Calling Web APIs lets the app connect to the outside world

What is a Web API?

- Web API = Web accessible Application Programming Interface accessible via HTTP to allow programmatic access to applications
  - Also known as Web Services
  - Can be accessed by a broad range of clients including browsers and mobile devices
- Web API is a web service that accepts requests and returns **structured data** (JSON in most cases)

  - Programmatically accessible at a particular URL
  - You can think of it as a Web page returning JSON instead of HTML
- Major goal = **interoperability between heterogeneous systems**

Web Services Principles



- **Resources have unique address** (**nouns**) i.e., a URI

Any **information that can be named can be a resource**: a document or image, a dynamic service to get weather or news, a collection of books and their authors, and so on

e.g., http://example.com/customers/123

- **Can use a Uniform Interface** (**verbs**) to access them:

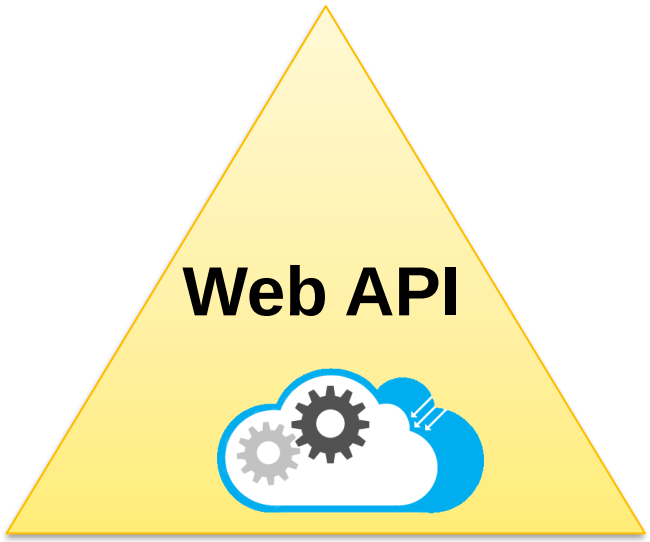
  - HTTP verbs: GET, POST, PUT, and DELETE

- **Resource has representation(s)** (**data format**)

  - A resource can be in a variety of data formats such as JSON and XML

Web API Main Concepts

Nouns (Resources)
e.g., http://example.com/employees/12345

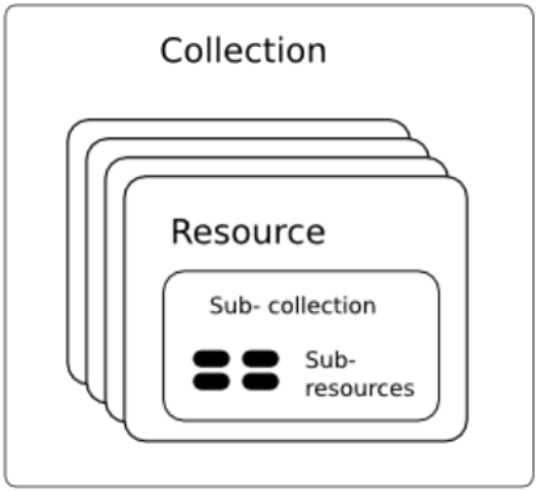


Verbs
e.g., GET, POST

Representations
e.g., XML, JSON

Naming Resources

- Web API uses URL to identify resources



Often **api** path is used for better organization

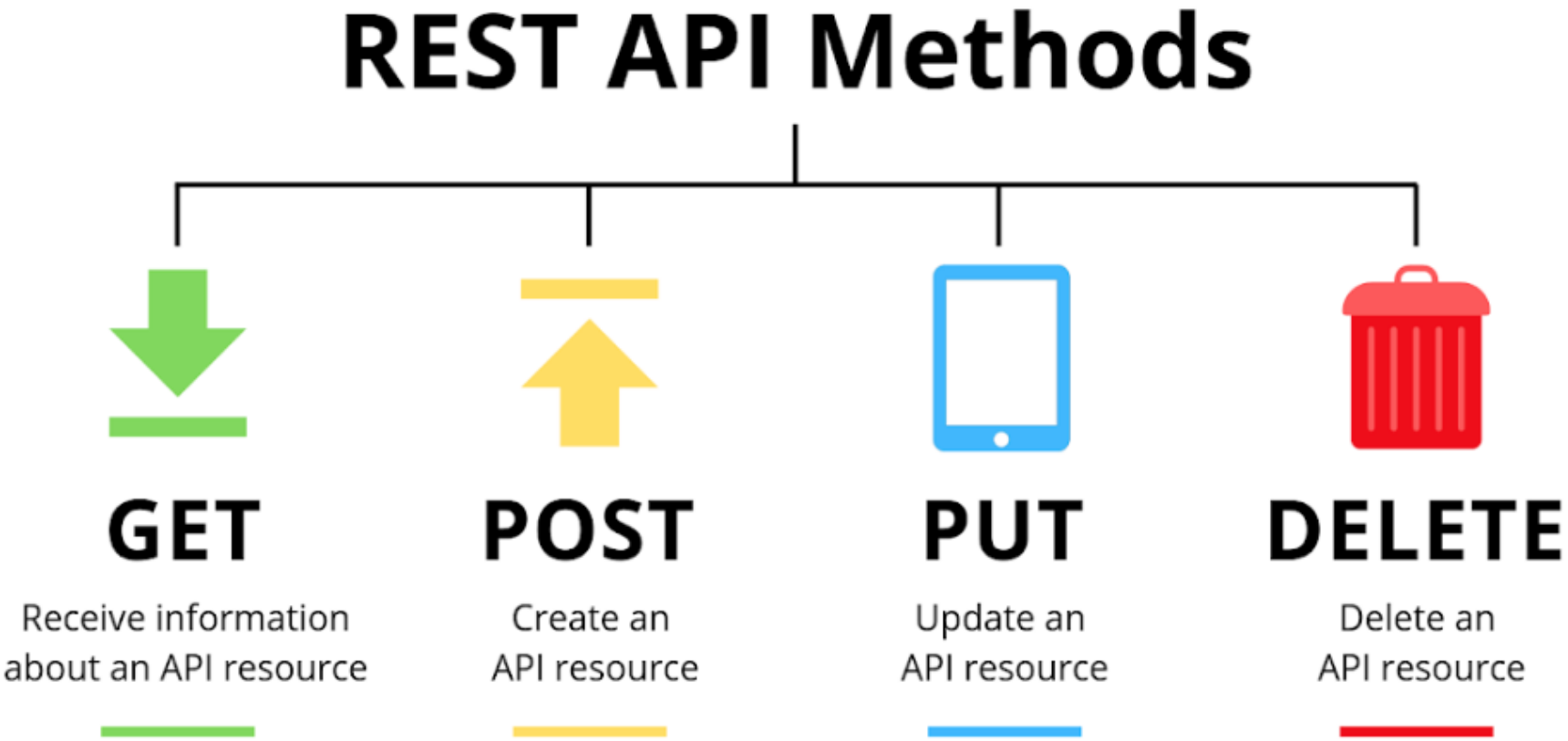
  - http://localhost/**api**/books/
  - http://localhost/api/books/ISBN-0011
  - http://localhost/api/books/ISBN-0011/authors

  - http://localhost/api/classes
  - http://localhost/api/classes/cmps356
  - http://localhost/api/classes/cs356/students

- As you traverse the **path** from more generic to more specific, you are navigating the data

HTTP Verbs

HTTP Verbs represent the **actions** to be performed on resources



CRUD (Create, Read, Update and Delete) Operations and their Mapping to HTTP Verbs

- **GET** - Read a resource
  - **GET** /books - Retrieve all books
  - **GET** /books/:id - Retrieve a particular book
- **POST** - Create a new resource
  - **POST** /books - Create a new book
- **PUT** - Update a resource
  - **PUT** /books/:id - Update a book
- **Delete** – Delete a resource
  - **DELETE** /books/:id - Delete a book

The resource data (e.g., book details) are placed in the **body** of the request

Example 2 - Task Service API

Task	Method	Path
Create a new task	POST	/tasks
Delete an existing task	DELETE	/tasks/{id}
Get a specific task	GET	/tasks/{id}
Search for tasks	GET	/tasks
Update an existing task	PUT	/tasks/{id}

Representations

- In all requests and responses, it is important to share data in a **format which both the client and server can understand**
- Two main formats are commonly used:
- **JSON**

```
{
    code: 'cmp312',
    name: 'Mobile App Development'
}
```

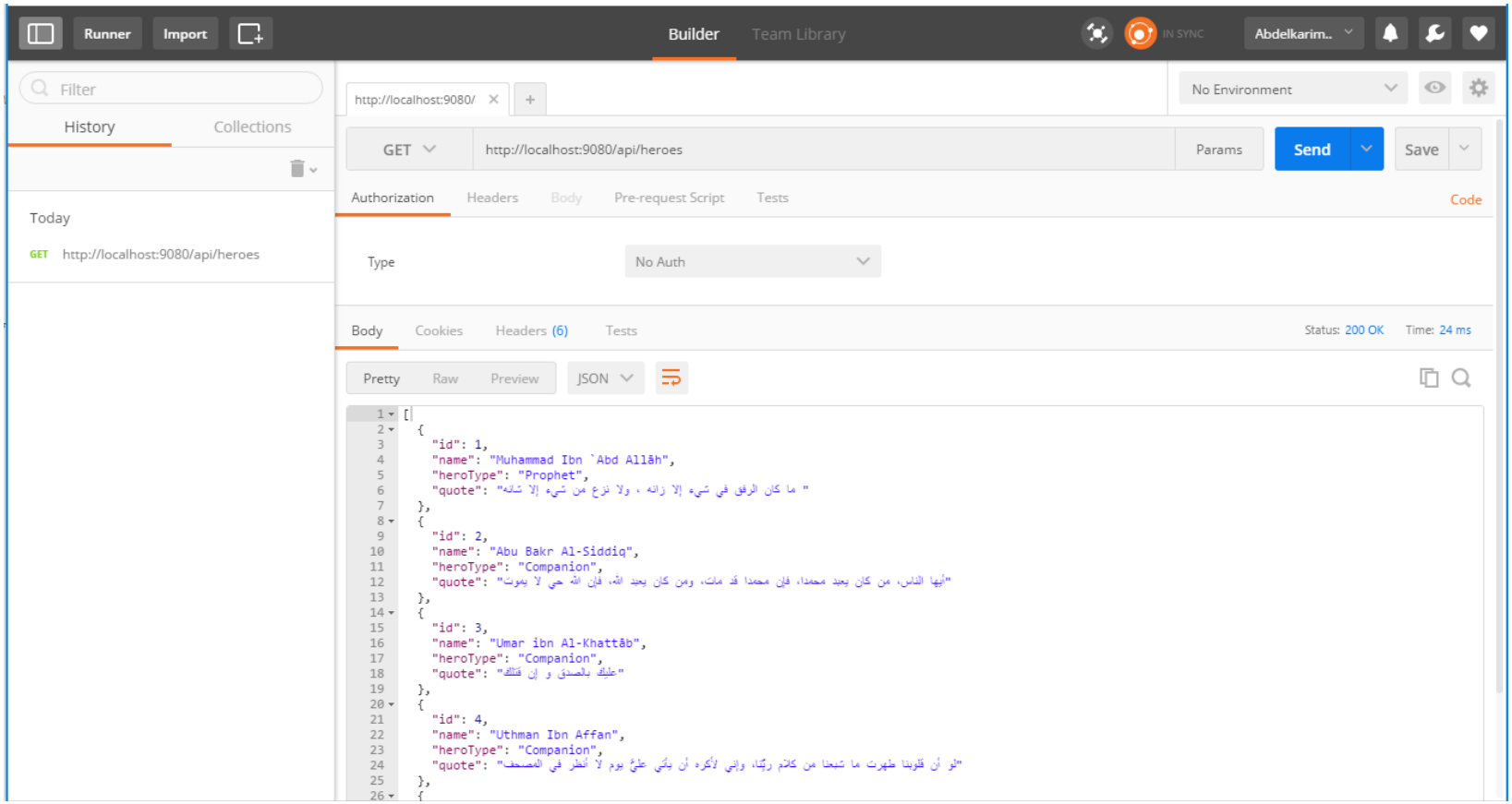
- **XML**

```
<course>
   <code>cmps312</code>
   <name>Mobile App Development</name>
</course>
```

Testing Web API

- Using Postman to test Web API

https://www.postman.com/downloads/

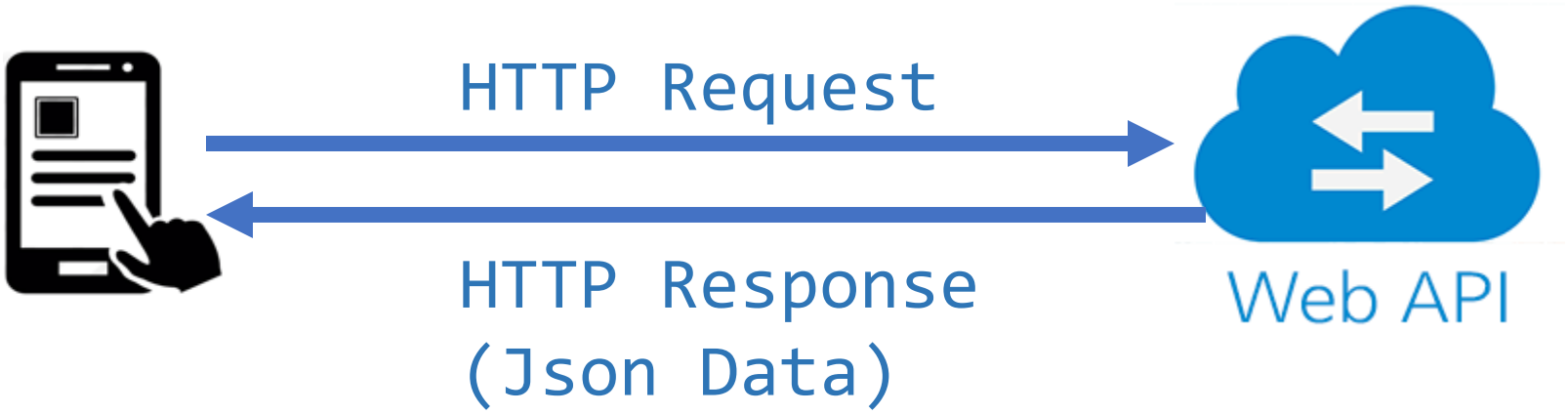


2

Ktor

Ktor Client

- **Ktor** provides HTTP client library for a mobile app to call a remote Web API
  - Make HTTP requests and handle responses



Ktor – 3 Programming Steps

1. Define **Serializable Data Classes** for input/output objects used when interacting with the Web API
2. Create a **Ktor client** and add the necessary plugins
3. Use the client **`.get`**, **`.post`**, **`.put`**, **`.delete`** methods to interact with the remote Web API

1. Define Serializable Data Classes for input/output objects used when interacting with the Web API

```
@Serializable
data class Country (
    // Map alpha3Code property in the json file
    // to the code property
    @SerialName("alpha3Code")
    val code: String = "",
    val name: String,
    val capital: String,
    @SerialName("region")
    val continent: String,
    @SerialName("subregion")
    val region: String,
    val population: Long,
    val area: Double = 0.0,
    val flag: String,
)
```

2. Ktor Client

- Create the client

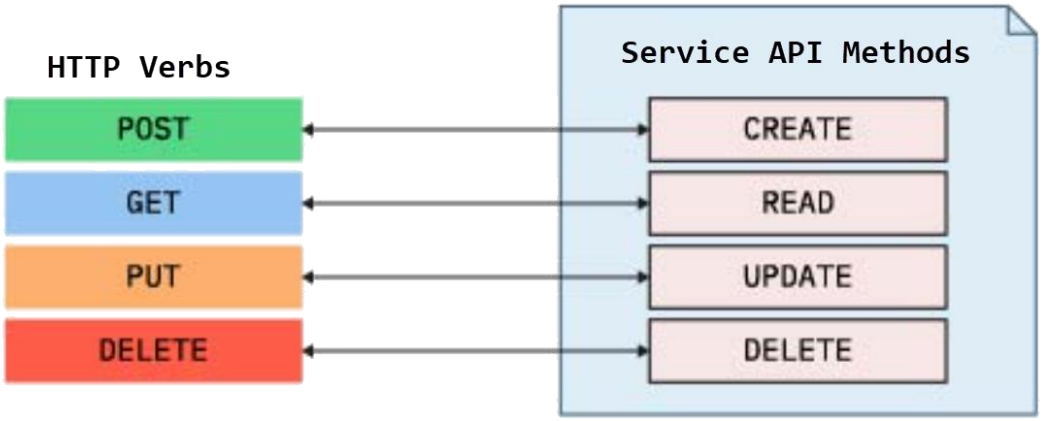
```
import io.ktor.client.*
val client = HttpClient()
```

- Add plugins to extend the client functionality, such JSON serialization, and Logging

```
val client = HttpClient() {
//Json Plugin auto-parse from/to json when sending and
receiving data from the Web API
    install(JsonFeature) {
        serializer = KotlinxSerializer()
    }
    //Log HTTP request/response details for debugging
    install(Logging) {
        level = LogLevel.ALL  // or .Headers or .Body
    }
}
```

3. Use Get/Post/Put/Delete to interact with the Web API

- HttpClient provides specific functions for basic HTTP methods: get, post, put, and delete.




```
const val BASE_URL = "https://api.polygon.io/v1/open-close"
val symbol = "Tesla"
val url = "$BASE_URL/$symbol"
println(">>> Debug: getStockQuote.url: $url")
val stockQuote  = client.get<StockQuote>(url)
```

Path Parameters vs. Query Parameters

- Required parameters can be passed using **path parameters** appended to the URL path
  - E.g., **/students/1234** this will return the details of the student with the id 1234
- Named **query parameters** can be added to the URL path after a **?** E.g., **/posts?sortBy=createdOnDate**
- Query parameters are often used for **optional** parameters (e.g., optionally specifying the property to be used to sort of results)

20

Post / Put Request

- Set the body of a request using body property
  - It accepts different types of payloads, including plain text or an object that get auto-serialized to a Json document

```
val response: HttpResponse = client.post("http://localhost:8080/posts") {
    body = "Body content"
}
```

```
val response: HttpResponse = client.post("http://localhost:8080/customers") {
    contentType(ContentType.Application.Json)
    body = Customer(3, "Ktor", "Client")
}
```

Delete Request

- Use the client.**`delete`** method to delete a resource
  - Specify the resource id to be deleted in the request url

```
val url = "https://jsonplaceholder.typicode.com/todos/1"
val response: HttpResponse = client.delete(url)

if (response.status == HttpStatusCode.OK) {
    // HTTP-200
}
```